# Методичка 7.3 - Применение и обход антиотладочных приемов. Часть 2.

**Использование специальных функций WinAPI**

Операционная система Windows предоставляет набор специальных функций, которые могут быть использованы для обнаружения того, что программа работает под отладчиком.

- IsDebuggerPresent
- CheckRemoteDebuggerPresent
- NtQueryInformationProcess

Функция IsDebuggerPresent показывает, запущен ли вызывающий ее процесс в контексте отладчика. Принцип работы этой функции мы разберем позже.

Функция CheckRemoteDebuggerPresent выясняет, отлаживается ли указанный процесс. Эта функция работает также, как и предыдущая, но она может проверить не только себя, но и любой существующий процесс в системе. Использование этой функции будет полезно, если программа имеет несколько процессов. Например, первый процесс программы выполняет основные функции программы, а второй периодически проверяет первый на наличие отладчика.

Функция NtQueryInformationProcess может быть использована для получения информации об указанном процессе. Принцип работы этой функции мы также разберем позже.

**Принцип работы специальных функций WinAPI**

Давайте попробуем разобраться, как работают эти функции. Это нам понадобится, чтобы понять, как их можно обойти.

Функции IsDebuggerPresent и CheckRemoteDebuggerPresent работают идентично. Разница лишь в том, что вторая может получить информацию по любому существующему процессу в системе, а не только для себя.

Данные функции выполняют чтение флага BeingDebugged из PEB структуры. PEB расшифровывается, как Process Environment Block, это некоторая структура в операционной системе Windows, которая содержит подробную информацию по процессу. Среди различных параметров, в этой структуре есть флаг BeingDebugged по значению которого можно определить отлаживается указанный процесс или нет. Если флаг выставлен в 1, то отлаживается, если выставлен в 0, то, соответственно, не отлаживается. Этот флаг выставляется операционной системой.

**Принцип работы функции NtQueryInformationProcess**

Теперь давайте разберемся, как работает функция NtQueryInformationProcess.

Функция принимает пять параметров. Первый параметр это дескриптор запрашиваемого процесса. Можно указать свой процесс, например, через функцию GetCurrentProcess(). Второй параметр - это константа (значение из объединения ProcessInformationClass), указывающая на данные, которые мы хотим получить из процесса.

Если в качестве второго параметра мы укажем 0x07, то получим ProcessDebugPort, если 0x1F, то ProcessDebugFlags, если 0x1E, то ProcessDebugObjectHandle.

Если какое-нибудь из этих значений выставлено, то процесс отлаживается.

Для того, чтобы продемонстрировать обнаружение отладчика в нашем примере мы будем использовать функцию IsDebuggerPresent.

*Исходный код программы prog-5.3.c*

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char* argv[]) {

  BOOL isDebugged = IsDebuggerPresent();

  if (isDebugged) {
    printf("\nDebugger has been detected!\n");
return 0;
  }

  printf("\nIt’s OK\n");
  return 0;
}
```

Компилируем программу.

```
> cl prog-5.3.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO
```

Запускаем программу:

```
> prog-5.3.exe

It’s OK!
```

Видим, что программа успешно отработала. Теперь давайте попробует запустить ее под отладчиком.

Запускаем отладчик OllyDbg, открываем программу prog-5.3.exe и запускаем ее - F9.

Debugger has been detected!

Видим, что отладчик был обнаружен и программа завершила свою работу.

Давайте попробуем обойти этот способ антиотладки.

Как вы помните, функция IsDebuggerPresent проверяет значение флага BeingDebugged в структуре PEB. Для того, чтобы эта проверка проходила успешно, нам необходимо изменить значение флага BeingDebugged в процессе отладки, а значит нам нужно знать, где расположена структура PEB в памяти.

Для получения адреса структуры PEB мы воспользуемся специальным плагином - StollyStructs для OllyDbg.

Рекомендации по установке плагина StollyStructs

Ссылка: https://tuts4you.com/download/1275/

Распаковывать архив и файлы StollyStruct.dll и StollyStructs.ini необходимо положить в директорию, где у нас установлен отладчик - C:\Program Files (x86)\odbg110.

Давайте запустим отладчик и откроем нашу программу. Поставим точку останова перед вызовом функции IsDebuggerPresent и продолжим выполнение программы - F9.

Теперь посмотрим содержимое структуры PEB.

Plugins - StollyStruct - Show PEB

Находим поле _PEB.BeingDebugged и видим, что его значение равно 1. Нам нужно его изменить на 0.

Edit - Binary - Edit - 0

Продолжаем выполнение программы - F9.

OK.

Видим, что программа успешно отработала и отладчик не был обнаружен. Мы успешно обошли проверку.

**Самоотладка (self-debug)**

Суть метода самоотладки заключается в том, чтобы попробовать отладить свой процесс.

Дело в том, что в ОС Windows процесс может отлаживаться только одним отладчиком в один момент времени. Суть проверки следующая. Если вас уже кто-то отлаживать, вы не сможете себя отладить.

Для реализации этого метода необходимо создать новый процесс (дочерний), и в нем использовать функцию DebugActiveProcess (PID) и передать ей PID родительского процесса. Если функция вернула ненулевое значение, значит программу никто в этот момент времени не отлаживает.

Для того, чтобы обойти этот метод необходимо либо заменить вызов функции DebugActiveProcess на NOP-инструкции, либо пропатчить функцию DebugActiveProcess, чтобы она возвращала всегда ненулевое значение, что свидетельствует об отсутствии отладки.

**Контроль целостности кода**

Контроль целостности кода является достаточно эффективным способом, но его очень редко используют, возможно, из-за сложности реализации.

Суть метода заключается в том, что программа читает некоторые области кода в адресном пространстве процесса, считает хэш от них и потом сравнивает полученных хэш с тем, что у нее был. Хэш-функция может быть любой, например, CRC32. Этот метод защищает программу от патчинга и от использования точек останова, которые мы ставим в отладчике, когда отлаживаем пошагово программу.

Давайте разберемся, как отладчик реализует точки останова.

Когда мы ставим точку останова, отладчик меняет первый байт инструкции на байт 0xCC. Байт 0xCC - это код специальной инструкции INT3. Когда процессор видит эту инструкция, срабатывает прерывание и выполнение кода приостанавливается.

Так поступают многие отладчики, и OllyDbg не является исключением. Получается, если мы ставим точку останова в коде мы вмешиваемся в код программы, а это значит, если этот код проверяется на контроль целостности, мы эту проверку не пройдем.

Для того, чтобы обойти эту проверку необходимо найти и убрать эту проверку, например, заменить инструкциями NOP.

**Использование специальных плагинов**

*ScyllaHide*

*https://github.com/x64dbg/ScyllaHide*

*HideOD*

*https://www.aldeid.com/wiki/OllyDbg/HideOD*

*HideDebugger*
*https://www.aldeid.com/wiki/OllyDbg/HideDebugger*

Крайне редко разработчики придумывают уникальные способы защиты своих программ от реверс-инжиниринга, как правило, это сводится к стандартным приемам, которые мы разобрали.

Конечно, то, что мы делали в ручном режиме, можно автоматизировать. И это уже сделано. Для того, чтобы свести потерю времени к минимум для обхода антиотладочных приемов, мы можем воспользоваться специальными плагинами.

Наиболее известные плагины для отладчика OllyDbg представлены на слайде.